Hindawi
Journal of Sensors
Volume 2017, Article ID 4209397, 11 pages
https://doi.org/10.1155/2017/4209397

Research Article

Parallel Algorithm for Wireless Data Compression and Encryption

Qin Jiancheng,[1] Lu Yiqin,[1] and Zhong Yu[2,3]

[1]*School of Electronic and Information Engineering, South China University of Technology, Guangdong, China*
[2]*School of Software, South China University of Technology, Guangdong, China*
[3]*China Telecom Co., Ltd., Zhaoqing Branch, Guangdong, China*

Correspondence should be addressed to Lu Yiqin; eeyqlu@scut.edu.cn

Received 4 October 2016; Revised 24 December 2016; Accepted 11 January 2017; Published 12 February 2017

Academic Editor: Fanli Meng

As the wireless network has limited bandwidth and insecure shared media, the data compression and encryption are very useful for the broadcasting transportation of big data in IoT (Internet of Things). However, the traditional techniques of compression and encryption are neither competent nor efficient. In order to solve this problem, this paper presents a combined parallel algorithm named "CZ algorithm" which can compress and encrypt the big data efficiently. CZ algorithm uses a parallel pipeline, mixes the coding of compression and encryption, and supports the data window up to 1 TB (or larger). Moreover, CZ algorithm can encrypt the big data as a chaotic cryptosystem which will not decrease the compression speed. Meanwhile, a shareware named "ComZip" is developed based on CZ algorithm. The experiment results show that ComZip in 64 b system can get better compression ratio than WinRAR and 7-zip, and it can be faster than 7-zip in the big data compression. In addition, ComZip encrypts the big data without extra consumption of computing resources.

1. Introduction

With the rapid expanding of IoT (Internet of Things), lots of sensors are available in various fields, and the volume of data in wireless networks is increasing in a marvelous speed. As more and more users share the same wireless channel, the bandwidth becomes rare. How to make full use of the wireless bandwidth is a problem.

Another problem is the security in wireless networks. As the signals broadcast in the air and every node can receive them, the data needs protection.

Data compression is a smart way to speed up the wireless network transportation, and data encryption can protect the transporting information. A practical way is using some high-performance nodes (named "wireless messenger" [1]) in the WSN (Wireless Sensor Network) to gather, store, and transmit the data of lightweight nodes. This way can avoid the bottleneck in the WSN caused by the weaknesses of the lightweight nodes: limited energy, short wireless distance, low performance, and so on.

But the practical problem still exists: when facing the GBs or TBs of big data in IoT, the traditional techniques of compression and encryption are neither competent nor efficient. For example, the software WinRAR has a small data window of 4 MB, which will limit the compression ratio. And its compression speed is not fast enough. Moreover, using AES (Advanced Encryption Standard) to encrypt the big data will consume the computing resources and decrease the speed.

We need to enhance the traditional techniques at the following aspects: higher speed, better compression ratio, and faster encryption. To achieve this goal, we may utilize some hopeful ways (e.g., a larger compressing data window and a parallel pipeline for compression and encryption).

Meanwhile, the exciting development of mobile hardware supports the proposed improvements. For example, the latest ARM (Advanced RISC Machines) platforms have multicore CPUs with low energy consumption, and current flash memory has enough capacity and well performance to store a large data window.

This paper proposes a combined parallel algorithm named "CZ algorithm" to compress and encrypt the big data efficiently. CZ algorithm has the following features:

(1) It uses the compression format named "CZ format" [2], which supports unlimited data window size.

(2) It mixes the coding of compression and encryption. The encryption named "CZ encryption" acts as a type of chaotic cryptosystem, which will not consume extra computing resources to slow down the compression.

(3) It uses the parallel pipeline named "CZ pipeline," which can use multiple threads to speed up the compression and encryption.

We upgrade our shareware named "ComZip" in the 64 b system with brand-new CZ algorithm. The newest ComZip supports a large data window up to 1 TB and mixes the compression and encryption together. Yet the data window is still limited by the actual size of RAM. As a contrast, another popular software 7-zip in the x64 platform supports the data window up to 4 GB, and it can use AES encryption.

We do some experiments to compare the efficiencies of processing the big data among ComZip, WinRAR, and 7-zip. The results show that ComZip can get better compression ratio than WinRAR and 7-zip, and it can be faster than 7-zip in the big data compression. Moreover, ComZip encrypts the big data without consuming extra computing resources.

In our experiment, limited by the computer memory of 32 GB, ComZip has a data window of only 4 GB. We believe that ComZip may have better compression ratio when the data window is larger. To make further experiments, we provide a 64 b version of ComZip in the website. The researchers may download it from http://www.28x28.com/doc/cz_x64.html.

The remainder of this paper is structured as follows.

Section 2 expresses the problems of big data compression and encryption in wireless communication. Section 3 introduces the parallel pipeline "CZ pipeline" and its framework. Section 4 describes the upgraded "CZ format" and "CZ algorithm." Section 5 describes the chaotic cryptosystem "CZ encryption." The experiment results are given in Section 6. The conclusions and future work are given in Section 7.

2. Problems of Big Data Compression and Encryption for Wireless Communication

Since the nodes are increasing in IoT and they share the wireless bandwidth, fast data transportation cannot depend on the high bandwidth assignment. Instead, we can use high-performance hardware to compress and encrypt the data for the transportation and security protection. But treating TBs or PBs of big data is a challenge. We are facing the following problems of big data compression and encryption.

(1) How to Encode the Big Data Fast Enough? A traditional "fast" technique may be slow in the field of big data. For example, if a serial compression algorithm can encode 10 MB in a second, then encoding 1 TB needs 29.1 hours. Speed is always important for the big data treatments.

Parallel computing is a way to solve this problem, but changing the serial algorithms is difficult. We have to try to divide the tasks and avoid the data relativity, which will prevent the parallel acceleration.

(2) How to Use a Large Data Window for Compression? We discussed the data window in [2] and revealed that a large data window might improve the compression ratio. But it is difficult to enlarge the data window, because the length of the index will grow, which will decrease the compression ratio.

We use multilevel indexes to solve this problem. And we have to upgrade the compression format to support a data window larger than 4 GB.

In this paper, we define the compression ratio as follows:

$$R = 1 - \frac{D_{\text{zip}}}{D}. \tag{1}$$

D_{zip} and D are the volumes of the compressed and original data, respectively. If the original data are not compressed, $R = 0$. If the compressed data are larger than the original data, $R < 0$. Always $R < 1$.

(3) How to Save the Computing Consumption of Encryption? Although AES algorithm is fast, it still consumes computing resources during the big data encryption. Since the data volume is considerable, this consumption will increase the energy cost and decrease the speed of data coding.

Chaotic encryption may be a way to save the consumption. We regard the lossless compression system as a chaotic cryptosystem. Thus the process of compression is also the process of encryption, and it does not need extra computing resources.

To solve the problems, we need to review the main related works around big data compression and encryption.

From the view of academic classification, current mathematic models and methods of lossless compression can be divided into the following 3 classes:

(1) The compression based on the probabilities and statistics (e.g., Huffman coding and arithmetic coding [3]). In this class, the PPM (Partial Prediction Match) algorithm [4] based on the Markov chain model has a good compression ratio.

(2) The compression based on the dictionary indexes (e.g., LZ77/LZSS algorithm [5] and LZ78/LZW algorithm). The compression models of LZ series have the advantage of speed.

(3) The compression based on the order and repeat of the symbols (e.g., run-length coding and BWT (Burrows-Wheeler Transform) [6] coding).

Current popular compression software are the comprehensive applications of the above basic theories. Each software combines different compression models and methods to achieve better effects. Table 1 lists the features of popular compression software and ComZip.

Table 1: Features of compression software.

Software	Format	Basic algorithms	Maximum data window size		Shortages
			Support	Current	
WinZip	Deflat	LZSS & Huffman	512 KB	512 KB	Small data window; low compression ratio; weak big data support
WinRAR	RAR	LZSS & Huffman	4 MB	4 MB	Small data window; low compression ratio; weak big data support
	PPMd	PPM	—	—	Good compression ratio for text data only; weak big data support
Bzip2	BZ2	BWT & Huffman	900 KB (block)	900 KB	Small BWT block; low compression ratio; weak big data support
7-zip	LZMA	LZSS & arithmetic	4 GB	1 GB	No multilevel coding for large data window; limited big data support
ComZip	CZ	LZ77 & arithmetic	1 TB—unlimited	0.5 TB	Need larger data window for higher compression ratio

There are other compression software (e.g., PAQ and WinUDA). They might have high compression ratio, but they are slow and unfit for big data.

As the trend of the lossless compression, the data window in the scope of MBs is already insufficient (e.g., WinRAR). We have seen the experiment results in [2]. We consider the future belongs to the efficient compression for big data, including the fast algorithms of LZ series and the large data window in the scope of GBs or more. And another researching hotspot is JSCAC (Joint Source Channel Arithmetic Codes) [7].

In the field of information security, to improve the encryption performance for big data, researchers use parallel computing with the aid of hardware (e.g., ASIC (Application Specific Integrated Circuit) [8], FPGA (Field-Programmable Gate Array) [9]). And a researching hotspot is the GPU (Graphic Processing Unit) acceleration (e.g., the GPU implements of AES [10] and RSA (Rivest-Shamir-Adleman algorithm) [11]).

Based on the chaotic cryptography [12], a chaotic cryptosystem has the advantages of the encryption speed and the key length, which is fit for big data. The researchers use 2 chaotic signals for the system synchronization and encryption [13] and use discrete methods in the chaotic system to improve the security [14] and present the implementation and synchronization of the chaotic cryptosystem [15]).

The problems of big data compression and encryption are so complex that no one can solve them in a short while. In this paper we focus on 3 points: the parallel coding pipeline, the large data window of compression, and the mixed encryption.

3. Framework of Parallel CZ Pipeline

The simplest parallel computing way for the big data compression is dividing the data directly and using multithreads to compress each part of the data separately. This algorithm is easy and fast, but the data window will also be divided into small pieces for the threads, which will decrease the compression ratio. For example, if 8 threads share a data window of 1 GB, then the actual data window for each thread is 128 MB.

CZ algorithm uses another way: a parallel coding pipeline. This way can maintain the data window size and keep the compression ratio, and it can also support multithreads and increase the speed.

Like the pipeline in a CPU (Central Processing Unit), CZ pipeline has several segments, which divide the serial workflow of compression/decompression into pieces. Each piece can burden a short part of the compression task. The threads run concurrently in the segments of CZ pipeline.

Figure 1 shows the framework of CZ encoding pipeline. As the reverse framework is CZ decoding pipeline, we only present the encoding part here.

The "BWT filter" is alternative. In this paper we do not use it. Hence the original data go straight into the "character buffer." The data window is in the character buffer. LZ77 algorithm runs mainly in the "string match unit." The "command buffer" stores the code-words of control instructions, characters, and length/index pairs. The "instruction parser" translates the command code-words into CZ format codes, and these codes go into different Markov chain models. The "arithmetic encoder" changes the CZ format flow into binary data and put them into the "binary buffer." Then the pipeline can output the compressed data.

To avoid the data relativity, each buffer in CZ pipeline has different zones for reading and writing. Hence the threads can read and write the buffers concurrently. A new design is the command code-word, which can increase the depth of CZ pipeline and simplify the work of the threads.

Table 2 shows the design of the command code-word. The command buffer is a 32 b data buffer; hence we design the 32 b code-words to cover the information of controls, characters, and length/index pairs.

4. Description of CZ Format and CZ Algorithm

We introduced the previous design of CZ format in [2], which can support the data window size up to 4 GB and keep the optimized compression ratio. The current CZ format is upgraded and supports the large data window of 1 TB. We still use multilevel coding for indexes and lengths, but the details are changed according to the repeated tests. In the current

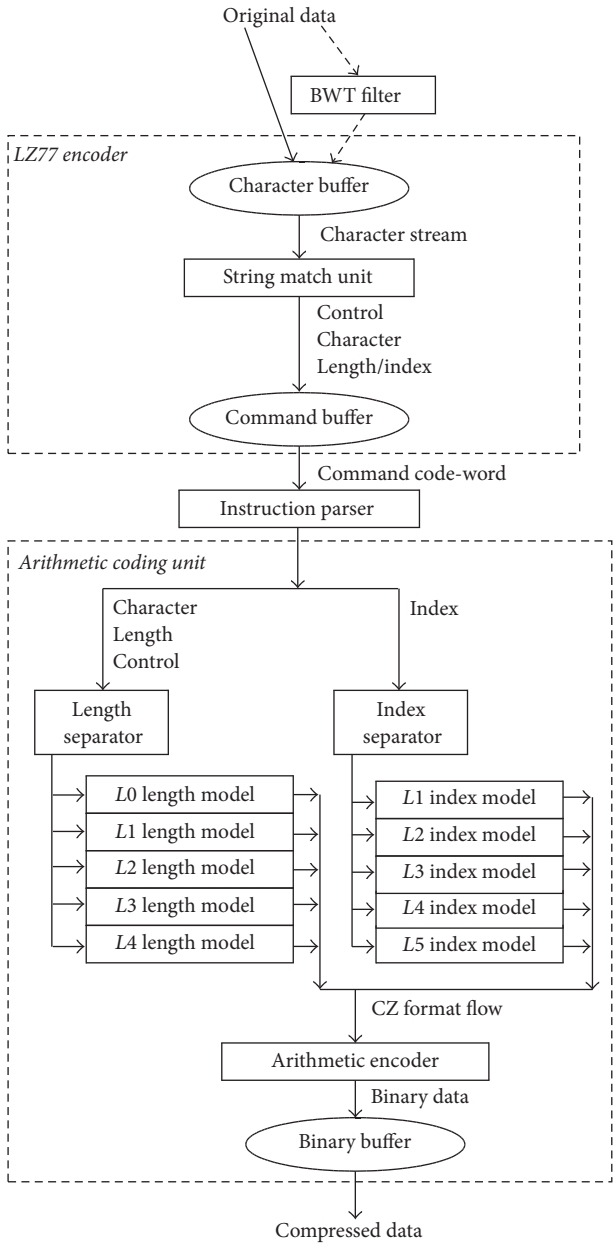

Figure 1: Framework of CZ encoding pipeline.

CZ format, we use a length separator code-word to lead the different code-words: characters, lengths, indexes, and so on.

We open the L_5 index to support the data window size up to 1 TB, and the L_6 index is ready for the future. We also open the L_3/L_4 length to support the string length up to 2^{64}. To simplify the treatments in CZ pipeline, we regard a single character as the L_0 length, which is a string with the length of 1. In Figure 1, the "L_0 length model" is the single character model.

Table 3 shows the design of the upgraded CZ format.

CZ algorithm is a combined parallel algorithm for multithreads, which is based on CZ pipeline. Figure 2 shows the main workflow of CZ algorithm.

To compress TBs or PBs of big data, the algorithm should be fast. CZ algorithm is based on LZ77 algorithm and self-adaptive arithmetic coding algorithm, and we have successfully optimized these 2 algorithms.

The traditional algorithm of self-adaptive arithmetic coding has the complexity of time $O(M)$. M is the amount of symbols. As a contrast, the optimized algorithm has the complexity of time $O(\mathrm{lb}M)$. The key reason is that the traditional algorithm uses a data structure of linked list to maintain and locate the statistic table for the Markov chain model, while the optimized algorithm uses a heap instead of the linked list.

The traditional LZ77 algorithm has the complexity of time $O(N^2)$. N is the data window size. As a contrast, the optimized LZ77 algorithm has the complexity of time $O(N)$. This is even better than the traditional LZSS algorithm, which has the complexity of time $O(N\mathrm{lb}N)$. The key reason is that the optimized LZ77 algorithm does not inspect all of the string matches. It inspects only a few hopeful matches and selects the best.

5. Description of CZ Encryption

CZ algorithm is not only a compression algorithm. We can use it as an encryption algorithm. The main idea is that we regard the lossless compression system as a chaotic system. If we use a key to change the initial state of the compression system, the decompression system cannot synchronize the state without this key.

Not all of the compression systems fit the usage of chaotic cryptosystems. For example, the algorithms of LZ series and Huffman coding have the minimal coding unit of 1 b, which is insecure because crackers have the chance to synchronize the system state in the middle of the compressed data stream.

According to the theory of arithmetic coding, we find that the self-adaptive arithmetic coding algorithm is exactly a kind of discrete implement for a chaotic cryptosystem. This chaotic cryptosystem is based on the shift-mapping model. The recurrence formula of this shift-mapping model is

$$x_{n+1} = kx_n \bmod 1 \quad (k > 1,\ 0 < x_n < 1,\ n = 1, 2, 3, \ldots). \tag{2}$$

x_n is a real number, and it stands for one of the states of a chaotic system. k is the system parameter. And the self-adaptive arithmetic coding algorithm uses the following formula to encode the data:

$$e_n = x_n d_n, \tag{3}$$

where d_n stands for the original data and e_n stands for the compressed data. Moreover, e_n can also stand for the encrypted data, if we use a key to change the initial system state. Because the arithmetic coding is a discrete implement and e_n, d_n are dynamic length integers, it is difficult to use the following formula to crack the encryption:

$$x^n = e_n^{-1} d_n. \tag{4}$$

Table 2: Design of command code-word.

Hex range (32 b)	Description	Next code-word(s)
0–ff	ASCII code of a character	None
100–10f	Control instruction	Control operand(s)
110–7fffffff	Reserved	
80000000–800001ff	Length of 0–511	Index code-word
80000200–80000237	Exponential length of 2^9–2^{64}	Index code-word
80000238–ffffffff	Reserved	
Index code-word		
0–7fffffff	Index below 2 GB	None
80000000	Reserved	
80000001–ffffffff	Index below 1 TB (high 31 b)	Index (low 32 b)

Table 3: Design of upgraded CZ format.

Hex range	Description	Next code-word
Length separator code-word (5 b):		
0–f	Single character (L_0 length, high 4 b)	Single character (low 4 b)
10–1a	Length of 1–11 (L_1 length)	Index code-word
1b–1e	Length of 12–75 (high 2 b)	L_2 length code-word (low 8 b)
1f	L_3 length identifier	L_3 length code-word (8 b)
L_3 length code-word (8 b):		
0–f9	Length of 76–325	None
fa	Control identifier	Control instruction (4 b)
fb–fe	Reserved	None
ff	L_4 length identifier	L_4 length code-word (16 b)
L_4 length code-word (8 b):		
0–30	Exponential length of 2^{16}–2^{64}	None
31–145	Reserved	None
146–ffff	Length of 325–65535	None
Index separator code-word (4 b):		
0	L_1 index (8 b), 256 B window	None
1	L_2 index (16 b), 64 KB window	None
2	L_3 index (24 b), 16 MB window	None
3	L_4 index (32 b), 4 GB window	None
4	L_5 index (40 b), 1 TB window	None
5–7	Reserved	None

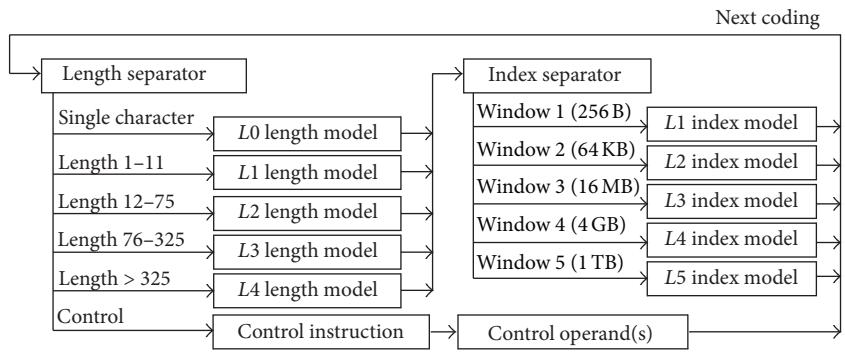

Figure 2: Main workflow of CZ algorithm.

```
function encrypt(K, D){   /* K is the key, D is the original data */
   R_salt = random(0–2^s);   /* s is the binary length of the random salt data, E.g. 128 b */
   init_state(0);   /* Initialize the system state */
   compress(K);   /* Compress the key, but do not output */
   output(compress(R_salt));   /* Compress the random salt data, and output */
   output(compress(D));   /* Compress the original data, and output */
}
function decrypt(K, E){   /* K is the key, E is the encrypted data */
   init_state(0);   /* Initialize the system state */
   compress(K);   /* Compress the key, but do not output */
   decompress(s);   /* Decompress salt data of length s, E.g. 128 b, and throw them */
   output(decompress(E));   /* Decompress the encrypted data, and output */
}
```

Algorithm 1

The core of CZ encryption is the above arithmetic coding, but CZ encryption is more complex, because the optimized LZ77 algorithm and multilevel index/length pairs are combined in the system. CZ encryption/decryption algorithm is shown in Algorithm 1.

By analyzing the above CZ encryption/decryption algorithm, we may see the following:

(1) If the decryption cannot get the correct key K, it will lose the chaotic system synchronization in the process *compress*(K). Then it cannot decompress the data E in the right routine. This is the security base of this chaotic cryptosystem.

(2) The most important time of the encryption is in the process *compress*(D). Compared to TBs or PBs of big data D, the key K and random salt data R_{salt} are so short that the compression time of K and R_{salt} can be ignored. Hence the encryption time is equal to the compression time. This implies that CZ encryption does not occupy extra time besides the compression time.

(3) The encryption and decryption support very long key, and the influence of the key length to the compression/encryption time can be ignored. For example, if we encrypt big data of 1 TB, the encryption time is approximate no matter whether the key length is 256 b or 20,000 b.

CZ encryption is a new coding method which combines the data compression and encryption. This is a valuable way for the efficient big data coding: it can save the consumption of computing resources, and it supports very long keys without influencing the performance.

We have designed a chaotic cryptosystem model named "CZ-Butterfly Effect Encryption Model" (CZ-BEEM), which combines CZ encryption, CZ algorithm, DHM (Diffie-Hellman-Merkle) or RSA algorithm, and LDPC (Low-Density Check-Parity Code) algorithm. CZ-BEEM can support a session key of 8,000 b or longer, and we can use a GPU to accelerate the DHM or RSA algorithm. The feasibility of CZ-BEEM is proved by an implement of this model. In the field of wireless communication, we have tested the speed of RSA in an embedded platform of STM32 in [16]. And we have tested the acceleration of LDPC decoding with a mobile phone GPU of PowerVR in [17]. However, to inspect the abilities of CZ-BEEM, we need more research works and experiment results. In this paper, we do not focus on CZ-BEEM.

CZ encryption is different from the simple congruent cipher. An important factor is that the output data of CZ encryption are compressed, which makes troubles to the attackers who want to distinguish the information confused and diffused in the bits [18]. The experiment results show that compressed data have good statistical characteristics: mostly randomness, unpredictability, and little redundancy. It infers that CZ encryption has reasonable strength against the cryptanalysis.

We analyze the difference between CZ encryption and a simple congruent cipher, focusing on their resistances to robust statistical attacks. If we change (3) a little, we can get an equation of a simple congruent cipher:

$$e_n = kd_n \mod m, \tag{5}$$

where k stands for the key. If we use FFT (Fast Fourier Transformation) to (5), we will get the equation of a classical congruent cipher as (6): Caesar cipher. Equations (5) and (6) are the same cipher system indeed:

$$e_n = k + d_n \mod m. \tag{6}$$

This encryption is weak because the length of e_n is a constant, and the attackers can easily split the output data into $e_1, e_2, \ldots$ to count them. If the original data $d_1, d_2, \ldots$ have some statistical information, the output data $e_1, e_2, \ldots$ will also have the similar information. For a simplified example, if the character "a" appears in the plain text at a ratio of 15%, the attackers can calculate the cipher text to find the slices at the same appearing ratio. These slices may stand for "a." This simple congruent cipher is insecure.

But this attacking method is unworkable to CZ encryption. The self-adaptive arithmetic coding algorithm is the main encoding unit of CZ encryption, which has the model of (3). If this model is ideal, the output data e_n will be an

Table 4: Original and compressed data file size (B).

File name	Description	Data file size (B)			
		Original	ComZip	WinRAR	7-zip
win7.iso	An ISO image file of installation DVD for Windows 7 (x64) Chinese version	3,341,268,992	2,480,098,840	3,100,386,508	2,701,623,708
vs2010.iso	An ISO image file of installation DVD for Virtual Studio 2010 Chinese version	2,685,982,720	2,561,514,632	2,626,046,379	2,588,193,704
ubuntu.vdi	A virtual machine image file of a Ubuntu Linux system	2,849,001,472	225,539,040	232,584,174	187,298,755
win2003.vdi	A virtual machine image file of a Windows 2003 system	2,835,402,752	1,205,393,648	1,299,062,724	1,030,057,340
lamp.vdi	A virtual machine image file of a Linux data partition	527,467,008	125,894,680	250,332,779	122,996,451
book.htm	A bookshop data file of storage records in HTML/XML format	346,499,594	4,627,200	4,312,296	5,464,344

endless real number (e.g., 0.35463865). It is a problem for the attackers to split this number into slices and calculate.

In practice, (3) has a discrete implement, which can change the real numbers into integers. But (3) and (5) are definitely different. In (5), each character may have a fixed length 8 b of the cipher text e_n, and the plain text "aaaaaaaa" will have 64 b. While in (3), e_n is compressed, and the plain text "aaaaaaaa" may have 0.003 b or 15.22 b or another length, which depends on the plain text. Even if the model is discrete, the arithmetic algorithm can encode a character with lower than 1 b. Moreover, each character has dynamic length of e_n (e.g., 0.003 b or 8.92 b or another value), and the length for the same character is also dynamic. Such compressed data are difficult to be split into slices to distinguish the original characters. On the other hand, if the attackers simply use a fixed length to split the data, the quantity of characters in a slice will be uncertain (e.g., 2.33 characters in 8 b, 0.971 characters, or another value). As the samples are unclear, the statistical analysis can hardly work.

The attackers may try the robust way: they can use very tiny slices and calculate. But it is never accurate enough to crack the arithmetic algorithm, and this robust method is inefficient. If a common user sets a key of 160 b in the CZ encryption system, the robust key-guessing attacks can search 2^{160} passwords to crack it, while the robust statistical attacks have to search much more than 2^{160} compressing system statuses. The robust key-guessing attacks are faster. Thus CZ encryption can prevent the robust statistical attacks.

CZ encryption is at least as strong as the chaotic cryptosystem of (2), which can generate a very long and mostly random key. Moreover, CZ encryption uses random salt data and LZ77 algorithm to enhance the security. In our experiment, we compare CZ encryption and a simple congruent cipher, and the results show the obvious difference between them.

6. Experiment Results

We have done some experiments to compare ComZip, WinZip, WinRAR, Bzip2, and 7-zip in [2]. The results indicate that WinZip and Bzip2 do not have good compression ratio because their data windows are too small (<1 MB); thus they do not have enough potential in the big data compression. In this paper we compare ComZip, WinRAR, and 7-zip in the experiments. The software versions are ComZip (x64) v20121221, WinRAR 4.20, and 7-zip (x64) 9.20. The operating system of this experiment platform is Windows. Although Linux and Android are popular in mobile systems, it is not easy to find a compression software for Linux/Android which has large data window to compare with ComZip.

We provide ComZip in the website. Researchers may use it to do more experiments with new data. It can be downloaded from http://www.28x28.com/doc/cz_x64.html.

The experiment platform is a common PC (Personal Computer) with the following equipment: AMD A8-5600K 4-core CPU, 32 GB DDR3 RAM, 1 TB 7200 rpm HDD (Hard Disk Driver), and Windows 7 (x64) Professional version. We expect that the future performance of high-end mobile platform such as 64 b ARM will keep up with this PC soon.

We can change the data window size of ComZip by modifying the parameter "cache" in the file "cz.ini." Odd values are also accepted (e.g., 371 MB and 1235 MB). But if the window size is too large, the system will have to use the virtual memory and the speed will be very slow. To avoid this, in our experiment, we use the window size 4096 − 4 = 4092 MB.

6.1. Comparison with the Effects of Compression and Encryption. In this experiment, we use the following data windows: ComZip 4092 MB, WinRAR 4 MB, and 7-zip 1024 MB. Table 4 shows sizes of the original and compressed data files. We select big files because the abilities of the big data window are limited by the size of the original data. For example, if a window of 4 GB compresses a single file of 4 MB, the actual useful window size is only 4 MB.

Table 5 shows the compression/encryption time of these files.

From Tables 4 and 5, we can find that the compression ratio of ComZip is better than WinRAR and 7-zip in most cases, and ComZip is faster than 7-zip. WinRAR is fast, but its compression ratio is limited by the window size of 4 MB. ComZip and 7-zip can also be faster by using a smaller data window.

Table 5: Compression and encryption time (seconds).

File name	Compression/encryption time (seconds)				
	ComZip	WinRAR	WinRAR & AES	7-zip	7-zip & AES
win7.iso	1059	350	379	1347	1359
vs2010.iso	1011	287	315	1097	1116
ubuntu.vdi	123	78	80	329	328
win2003.vdi	477	223	236	1035	1038
lamp.vdi	65	37	40	76	77
book.htm	19	9	9	32	32

Table 6: Compressed file size comparison (B).

Data window	ComZip	WinRAR	7-zip
1 MB	3,128,105,488	3,119,881,358	3,140,789,120
2 MB	3,116,960,272	3,108,416,160	3,128,772,615
4 MB	3,108,963,896	3,100,386,508	3,119,931,666
8 MB	3,104,976,200	—	3,114,504,857
16 MB	3,100,364,328	—	3,108,022,346
32 MB	3,092,400,272	—	3,100,097,611
64 MB	3,033,296,296	—	3,040,832,184
128 MB	2,984,543,048	—	2,991,642,706
256 MB	2,976,189,160	—	2,982,622,308
512 MB	2,918,815,872	—	2,924,014,236
1024 MB	2,697,174,696	—	2,701,623,708
2048 MB	2,481,901,456	—	—
4096 − 4 MB	2,480,098,840	—	—

The encryption time is connected to the compression process. The compression time of ComZip has already combined the time of CZ encryption. WinRAR and 7-zip use AES encryption. The difference of the time is not obvious for 7-zip with or without AES. We can even observe a case (ubuntu.vdi) that 7-zip with AES is faster than the compression alone. We consider the reason is that 7-zip also uses multithreads, which may save the time, but AES encryption still consumes the computing resources in one of the threads. As a contrast, CZ encryption does not consume extra resources.

6.2. Comparison with the Compression Ratio and Time. In this experiment, we use different data windows to compress the same original file named "win7.iso." Table 6 and Figure 3 show the relationship of the compressed file size and the data window size. From (1), we can find that this relationship is virtually the relationship of the compression ratio and the window size.

Table 7 and Figure 4 show the relationship of the compression time and the data window size.

From Table 6 and Figure 3, we find that the curves of compressed file size are very close, which indicates that when ComZip, WinRAR, and 7-zip use the same data window size, the difference of the compression ratio is not obvious. Generally, ComZip beats WinRAR and 7-zip in the compression ratio with the advantages of a larger data window and the upgraded CZ format.

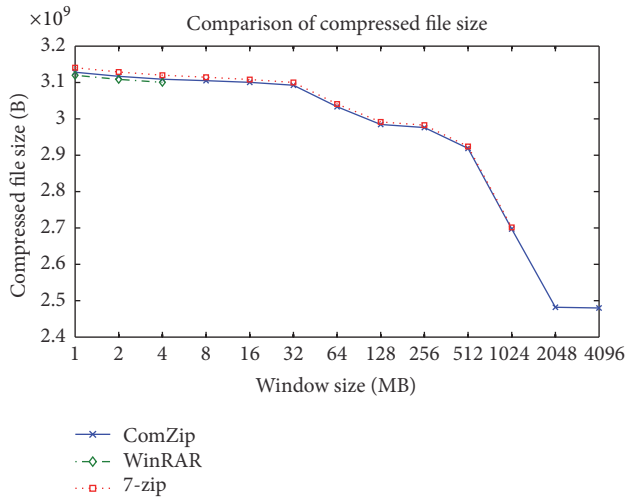

Figure 3: Compressed file size and data window size.

Compared to our previous experiments in [2], the new version of WinRAR is optimized in the speed. We find that WinRAR 3.71 uses a single thread, while WinRAR 4.20 uses multithreads. The latter is faster, but its data window is still limited in 4 MB; thus it cannot use a larger window for the big data to get better compression ratio.

We have predicted in [2] that ComZip would be faster than 7-zip when the data window exceeds 64 MB, and now we

Table 7: Compressing time comparison (seconds).

Data window	ComZip	WinRAR	7-zip
1 MB	654	431	535
2 MB	668	347	568
4 MB	677	350	591
8 MB	711	—	607
16 MB	768	—	634
32 MB	787	—	723
64 MB	817	—	1,209
128 MB	873	—	1,346
256 MB	960	—	1,411
512 MB	1,043	—	1,391
1024 MB	1,068	—	1,347
2048 MB	1,058	—	—
4096 − 4 MB	1,059	—	—

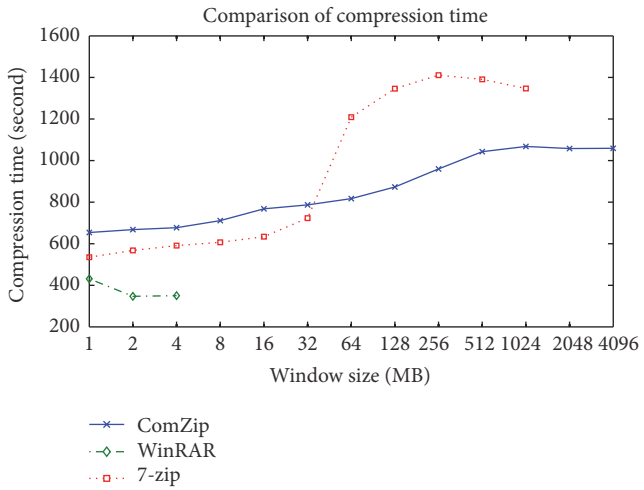

Figure 4: Compression time and data window size.

observe this case in Figure 4. We believe the main reasons are CZ pipeline and the optimized LZ77 algorithm, which has the complexity of time O(N). Moreover, Figure 4 supports that the time increment of ComZip is steady when the window size increases.

In Table 7 and Figure 4, ComZip is not faster than WinRAR and 7-zip when the data window is reduced to 4 MB or smaller. A probable reason is that ComZip is written by compatible C/C++ without some optimization in assembly language or machine language, which may influence the performance. But as the data window becomes larger to fit the big data, the performance of ComZip keeps up with that of 7-zip, while WinRAR's data window is too small to compete. Moreover, we can use the special hardware acceleration if it is necessary.

6.3. Exhibition with the Effects of CZ Chaotic Encryption. In this experiment, we use ComZip to compress and encrypt an image file with 256 gray scales and observe the effects of

Figure 5: Original image.

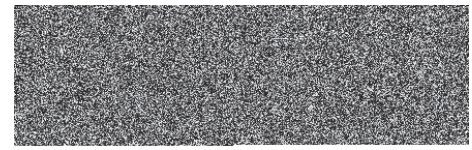

Figure 6: CZ-encrypted data image.

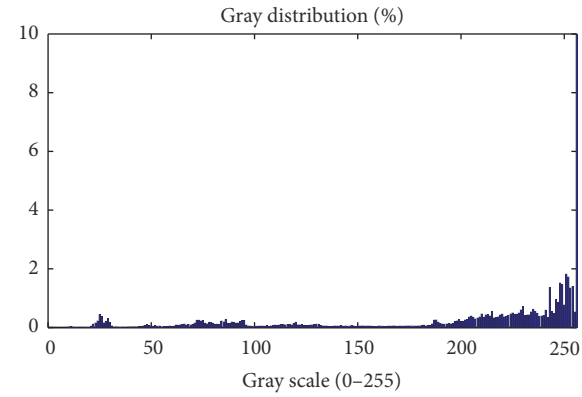

Figure 7: Gray distribution histogram of original image.

the encryption. Figure 5 shows the original image, which is a BMP (Bitmap) file of 106,680 B. Figure 6 shows the encrypted data, which are put into another image file with 256 gray scales: a BMP file of 62,680 B. From Figure 6 we cannot find any signal of Figure 5. And the pixels of Figure 6 are less than Figure 5, because the data are compressed.

Figures 7 and 8 are histograms. Figure 7 shows the gray distribution of Figure 5. As a contrast, Figure 8 shows the gray distribution of Figure 6, which has the statistic character of random distribution. This indicates that CZ encryption can defend the statistical attacks.

Because of the random salt data R_{salt} in CZ encryption, ComZip outputs random compressed data. If ComZip compresses the same data twice or more, we will get different outputs. To compare the outputs, we pick 2 segments with each length of 500 B from the encrypted data. Each segment is picked from the same position of the encrypted data, and the encryption key is the same, but R_{salt} is random. Figure 9 shows the different values of the ASCII codes for every character in the 2 segments. This indicates that CZ encryption has good randomness and the encrypted data are unpredictable, which can protect the information security.

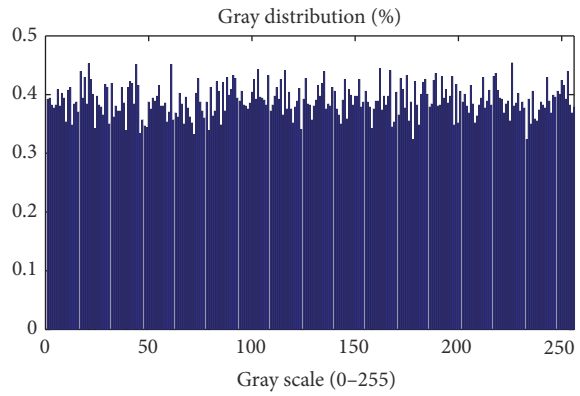

Figure 8: Gray distribution histogram of CZ-encrypted image.

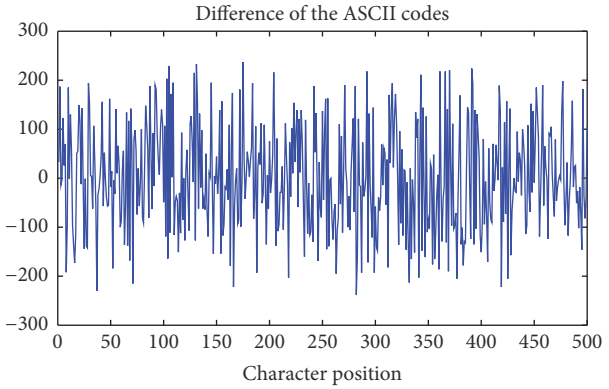

Figure 9: Difference between 2 CZ-encryptions.

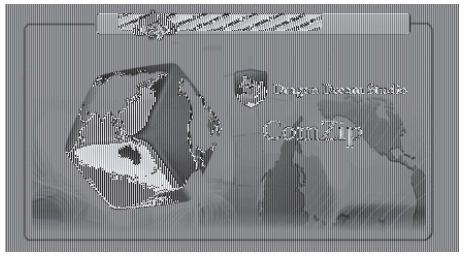

Figure 10: Caesar-encrypted data image.

6.4. Comparison with CZ Encryption and a Simple Congruent Cipher. In this experiment, we compare CZ encryption and a simple congruent cipher to show the difference between them. The simple congruent cipher is Caesar encryption, a classical but insecure cipher with the model of (6).

We use the same original data as Figure 5 to test CZ encryption and Caesar encryption. The results of CZ encryption are shown in Figures 6 and 7. By comparison, Figure 10 shows the Caesar-encrypted data, and Figure 11 shows the gray distribution of Figure 10.

It is well known that Caesar encryption is insecure, and now the results support it intuitively: we can recognize that Figure 10 is like Figure 5, and Figure 11 has similar statistic characters to Figure 7. Undoubtedly Caesar encryption is too weak to prevent the robust statistical attacks.

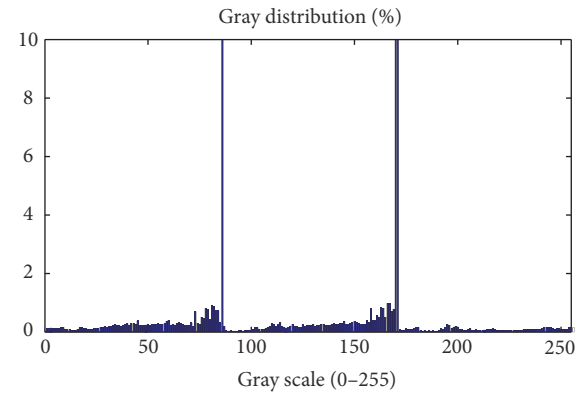

Figure 11: Gray distribution histogram of Caesar-encrypted image.

We can find the obvious difference between CZ encryption and Caesar encryption: Figure 6 versus Figure 10 plus Figure 7 versus Figure 11. The results show that CZ encryption has good statistical characters to prevent the attacks. CZ encryption is beyond a simple congruent cipher: it combines compression with encryption and becomes a chaotic cryptosystem.

From all of the experiment results in Section 6, we can get some support about the advantages of CZ algorithm: the compression ratio, compression/encryption time, the chaotic encryption effects, and so on.

Yet these results cannot completely reveal the latent abilities of CZ algorithm. We should do more research works for this algorithm (e.g., parallel optimization, hardware acceleration, and analysis of chaotic encryption). Thus we will gather more experiment results and upgrade ComZip.

7. Conclusions and Future Work

As the wireless network has limited bandwidth and insecure shared media, the data compression and encryption are very useful for the broadcasting transportation of big data in IoT. A way to avoid the bottleneck in the WSN is using high-performance nodes named "wireless messenger" to gather, store, and transmit the data of lightweight nodes. But the main problems are about high coding performance, big window size, and low computing consumption. Facing TBs or PBs of big data, traditional coding techniques expose their limitations of abilities.

Solving these problems is difficult. Enlarging the data window directly cannot get better compression ratio, because a larger window needs longer index code-words, which decreases the compression ratio. And a larger window needs more computing consumption to find the substring matches. A parallel algorithm has to avoid the conflicts of data access. Moreover, common encryption also causes the computing consumption.

This paper presents CZ algorithm, a combined algorithm for data compression and encryption. CZ algorithm uses CZ pipeline, new designed CZ format, and mixed compression/encryption coding. It is designed for the big

data treatment. A shareware ComZip is developed with CZ algorithm.

The experiment results indicate the advantages of CZ algorithm: the compression ratio of ComZip is better than WinRAR and 7-zip in most cases. The speed of ComZip is faster than 7-zip. The chaotic encryption of ComZip is feasible, and it does not consume extra computing resources, despite its key length. We believe these advantages are mainly from the parallel pipeline, the multilevel index/length codewords, and the chaotic cryptosystem.

Moreover, from the tables and curves, we get some support to predict that the trend of CZ algorithm may be better: ComZip can support a data window more than 4 GB, but currently it is limited by the memory capacity. As the window size increases, its compression/encryption time will increase steadily and slowly, which is a good character for the big data coding. The chaotic cryptosystem can support a key over 10,000 b without performance loss. And this discrete chaotic cryptosystem may have latent abilities to defend multiple attacks, but we need further experiments and analysis. We will continue the research.

Another future work is to make the program of CZ algorithm compatible to Linux/Android/Contiki platforms in order to fit more wireless network nodes. To support large data window, we may use flash memory or other new memory types.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. C. Qin and B. Sun, "Design of messenger oriented P2P framework in mobile Ad-hoc networks," in *Proceedings of the 11th International Conference on Advanced Communication Technology (ICACT '09)*, pp. 1101–1105, Gangwon-Do, Republic of Korea, 2009.

[2] J.-C. Qin and Z.-Y. Bai, "Design of new format for mass data compression," *Journal of China Universities of Posts and Telecommunications*, vol. 18, no. 1, pp. 121–128, 2011.

[3] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256–294, 1998.

[4] A. Moffat, "Implementing the PPM data compression scheme," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1917–1921, 1990.

[5] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[6] M. Burrows and D. J. Wheeler, *A Block-Sorting Lossless Data Compression Algorithm*, DIGITAL System Research Center, 1994.

[7] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and I. H. Witten, "Integrating error detection into arithmetic coding," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 1–3, 1997.

[8] A. Hodjat, D. Hwang, B. Lai et al., "A 3.84 Gbits/s AES crypto coprocessor with modes of operation in a 0.18-um CMOS Technology," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI (GLSVLSI '05)*, pp. 60–63, Chicago, Ill, USA, April 2005.

[9] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, pp. 545–557, 2001.

[10] S. A. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," in *Proceedings of the IEEE International Conference on Signal Processing and Communications (ICSPC '07)*, pp. 65–68, Dubai, United Arab Emirates, November 2007.

[11] A. Moss, D. Page, and N. Smart, "Toward acceleration of RSA using 3D graphics hardware," in *Proceedings of the 11th IMA International Conference on Cryptography and Coding*, pp. 369–388, December 2007.

[12] L. Kocarev, "Chaos-based cryptography: a brief overview," *IEEE Circuits and Systems Magazine*, vol. 1, no. 3, pp. 6–21, 2001.

[13] T. Yang, C. W. Wu, and L. O. Chua, "Cryptography based on chaotic systems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 44, no. 5, pp. 469–472, 1997.

[14] S. H. Wang, J. Y. Kuang, J. H. Li, Y. Luo, H. Lu, and G. Hu, "Chaos-based secure communications in a large community," *Physical Review E*, vol. 66, no. 6, Article ID 065202, 4 pages, 2002.

[15] M. Long, F. Peng, S. S. Qiu, and Y. F. Chen, "Implementation of a new chaotic encryption system and synchronization," *Journal of Systems Engineering and Electronics*, vol. 17, no. 1, pp. 43–47, 2006.

[16] Y. Lu, J. Zhai, R. Zhu, and J. Qin, "Study of wireless authentication center with mixed encryption in WSN," *Journal of Sensors*, vol. 2016, Article ID 9297562, 7 pages, 2016.

[17] Y. Lu, W. Su, and J. Qin, "LDPC decoding on GPU for mobile device," *Mobile Information Systems*, vol. 2016, Article ID 7048482, 6 pages, 2016.

[18] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.